3分決定グラフを用いた積項集合表現(アルゴリズムと計算量理論)

AUTHOR(S):

Yasuoka, Koichi

CITATION:

Yasuoka, Koichi. 3分決定グラフを用いた積項集合表現(アルゴリズムと計算量理論). 数理解析研究所講究録 1995, 906: 186-195

ISSUE DATE:

1995-04

URL:

http://hdl.handle.net/2433/59444

数理解析研究所講究録
906巻 1995年 186-195

3分決定グラフを用いた積項集合表現

京都大学大型計算機センター
安岡孝一*

1995年2月2日 (木)

Abstract

This paper presents Ternary Decision Diagrams which represent sets of products. This paper also presents manipulating methods for sum-of-products forms and ringsum-of-products forms using Ternary Decision Diagrams.

1 Introduction

Binary Decision Diagrams [1] are very efficient in representing and manipulating logic functions [2]. BDDs have become very popular in the field of VLSI logic system designs, and have occupied very high position at implementing CAD systems [3, 4, 8]. Nowadays BDDs are used to represent sets of products [5, 7] as well as logic functions. But BDDs are rather powerless to represent sets with binate literals, so new idea to represent sets of products is required.

Ternary Decision Diagrams were first introduced to represent Pseudo Kronecker Expressions [6], which are subclass of General Reed-Muller Expressions. More general in this paper, Ternary Decision Diagrams are presented as a graph representation of sets of products. The manipulation of sum-of-products forms and ringsum-of-products forms using TDDs is also presented.

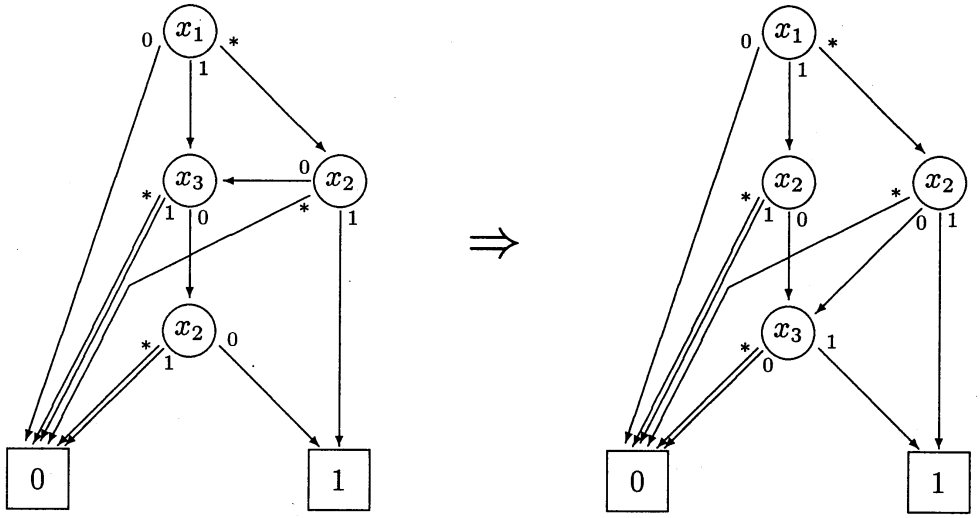
2 Ternary Decision Diagrams

Ternary Decision Diagrams are a direct 3-degree acyclic graph denoted by an 8-tuple $\langle V,\boxed{0},\boxed{1},\epsilon_0,\epsilon_1,\epsilon_*,X,\lambda\rangle$; where $\boxed{0}$ and $\boxed{1}$ are leaf nodes, V is a set of nodes except for the leaf nodes, ϵ_0, ϵ_1, and ϵ_* denote edges that are mappings from V to $V\cup\{\boxed{0},\boxed{1}\}$, $X\equiv\{x_1,x_2,\cdots,x_n\}$ is a set of variables, λ which represents labels on the non-leaf nodes is a mapping from V to X; and TDDs must satisfy:

i) **Variable Ordering Rule** (see Figure 1.)
There exists a total order $x_{\theta_1} > x_{\theta_2} > \cdots > x_{\theta_n}$ on X such that

$$\forall v\in V,\ \begin{cases} \epsilon_0(v)\in V \Rightarrow \lambda(v)>\lambda(\epsilon_0(v)) \\ \epsilon_1(v)\in V \Rightarrow \lambda(v)>\lambda(\epsilon_1(v)) \\ \epsilon_*(v)\in V \Rightarrow \lambda(v)>\lambda(\epsilon_*(v)) \end{cases} .$$

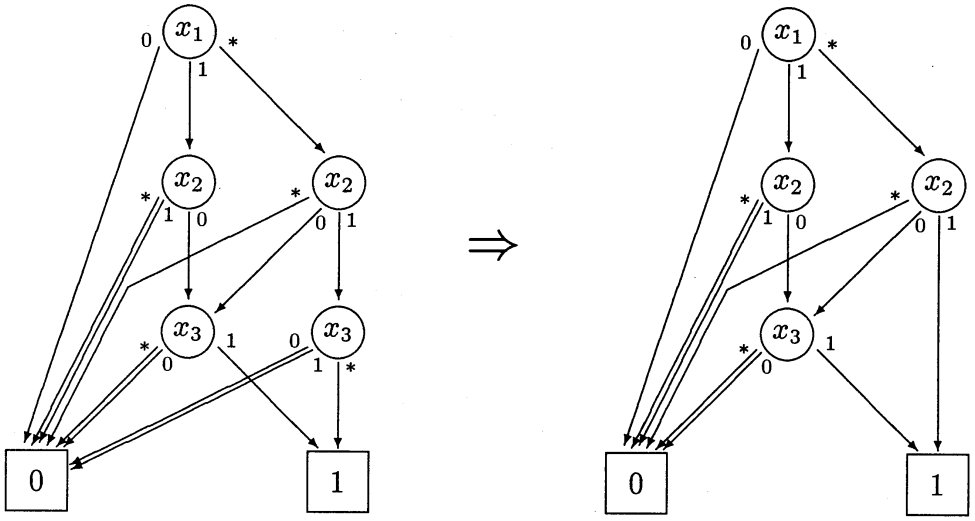
*Koichi Yasuoka, Kyoto University Data Processing Center Research and Development Division



(a) incorrect TDDs (b) correct TDDs

Figure 1: Variable Ordering Rule

ii) **Node Reduction Rule** (see Figure 2.)
$^{\forall}v \in V,\ \epsilon_0(v) \neq \boxed{0} \vee \epsilon_1(v) \neq \boxed{0}$.



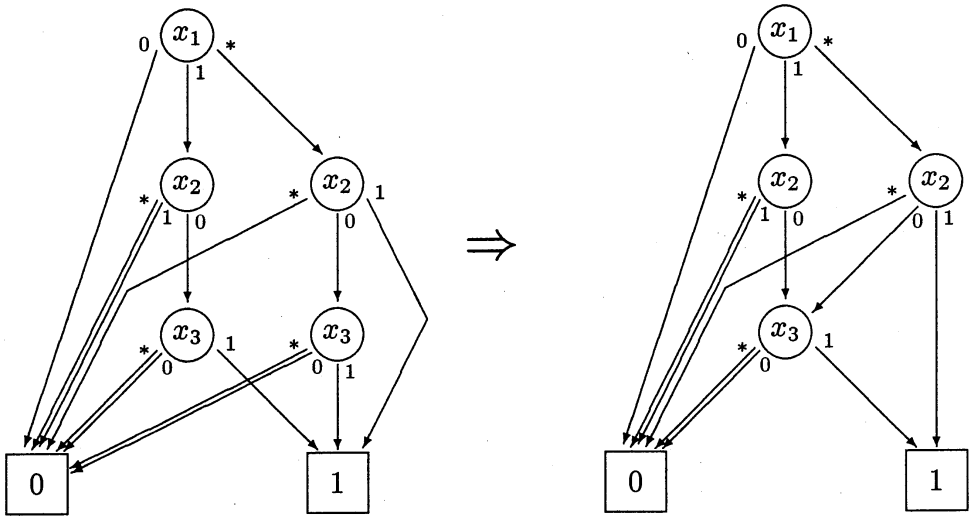
(a) incorrect TDDs (b) correct TDDs

Figure 2: Node Reduction Rule

iii) **Node Unification Rule** (see Figure 3.)
$^{\forall}v,v' \in V,\ \lambda(v)=\lambda(v') \wedge \epsilon_0(v)=\epsilon_0(v') \wedge \epsilon_1(v)=\epsilon_1(v') \wedge \epsilon_*(v)=\epsilon_*(v') \Rightarrow v=v'$.

We define a set of products represented by a node in TDDs as follows:

- $\boxed{0}$ represents the empty set $\emptyset$.
- $\boxed{1}$ represents the set $\{1\}$.



(a) incorrect TDDs (b) correct TDDs

Figure 3: Node Unification Rule

- A non-leaf node v∈V represents the union of the following three sets; the logical products of $\overline{\lambda(v)}$ and the set represented by $\epsilon_0(v)$, the logical products of $\lambda(v)$ and the set represented by $\epsilon_1(v)$, and the set represented by $\epsilon_*(v)$.

Figure 4 shows an example of sets of products represented by nodes in TDDs.

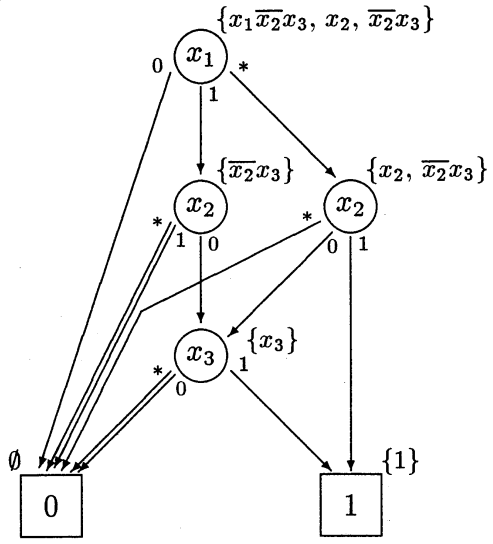


Figure 4: Sets of products represented in TDDs

3 Operations on TDDs

When a set of products is given, it can be regarded either as a sum-of-products form or as a ringsum-of-products form. For example, the set $\{x_1\overline{x_2}x_3,\ x_2,\ \overline{x_2}x_3\}$ can be regarded either as $x_1\overline{x_2}x_3+x_2+\overline{x_2}x_3$ or as $x_1\overline{x_2}x_3\oplus x_2\oplus\overline{x_2}x_3$. Hence, when we represent a set of products in

TDDs, we carry out different operations on TDDs according as our view of the set, namely, the sum-of-products form or the ringsum-of-products form.

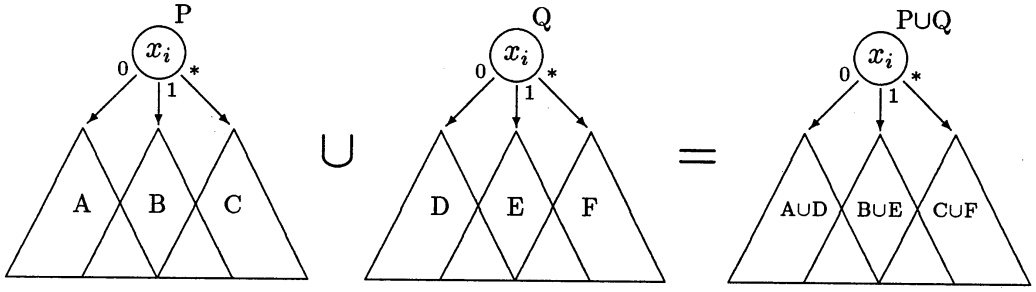
In the rest of this paper, we think of $\{\overline{x_i}\}$, $\{x_i\}$ ($x_i \in X$), $\emptyset$, and $\{1\}$ as basic sets that are given in TDDs from the beginning.

3.1 Manipulating sum-of-products forms on TDDs

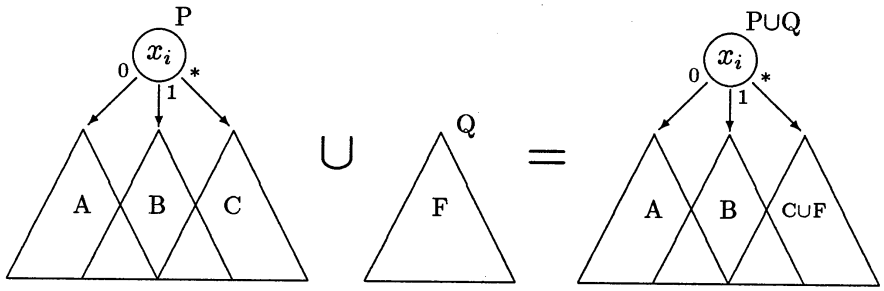
We consider essential operations on TDDs, in which we represent sum-of-products forms, in this section, namely, Union, Cartesian Product, Weak Division, Intersection, and Difference.

i) **Union**

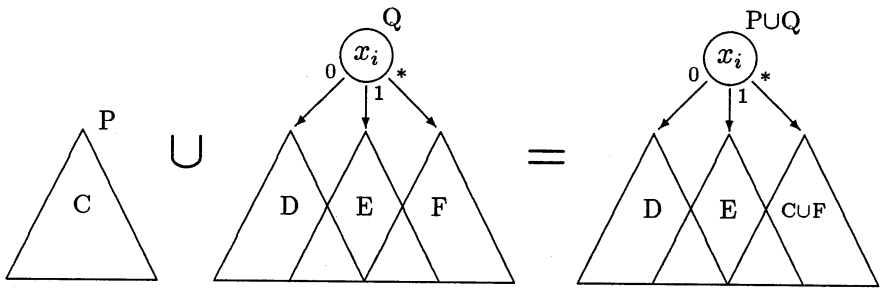
The set operator $\cup$ is used to make the sum of two sum-of-products forms. For example, $\{x_1\overline{x_2}x_3,\ x_2,\ \overline{x_2}x_3\}\cup\{x_2,\ \overline{x_1}x_3\}=\{x_1\overline{x_2}x_3,\ x_2,\ \overline{x_2}x_3,\ \overline{x_1}x_3\}$ can be regarded as $(x_1\overline{x_2}x_3+x_2+\overline{x_2}x_3)+(x_2+\overline{x_1}x_3)=x_1\overline{x_2}x_3+x_2+\overline{x_2}x_3+\overline{x_1}x_3$. Figure 5 shows the procedure for the operator $\cup$ on TDDs. The procedure terminates when it reaches to the leaf nodes, where the rules $P\cup\emptyset=P$, $\emptyset\cup Q=Q$, and $\{1\}\cup\{1\}=\{1\}$ are used.



(a) λ(node of P)=λ(node of Q)



(b) λ(node of P)>λ(node of Q)



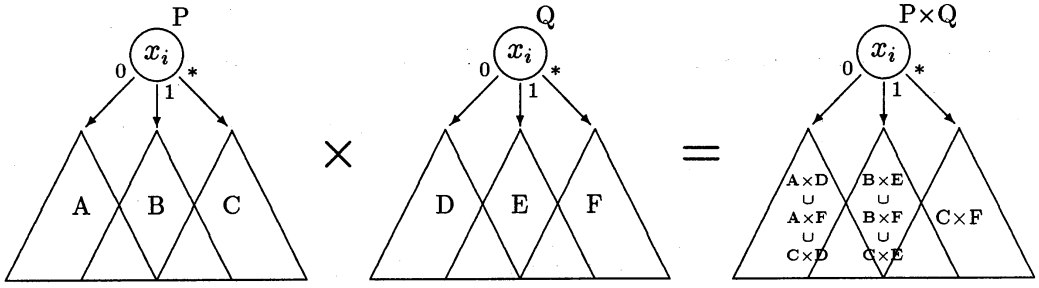
(c) λ(node of P)<λ(node of Q)

Figure 5: Procedure to compute PUQ

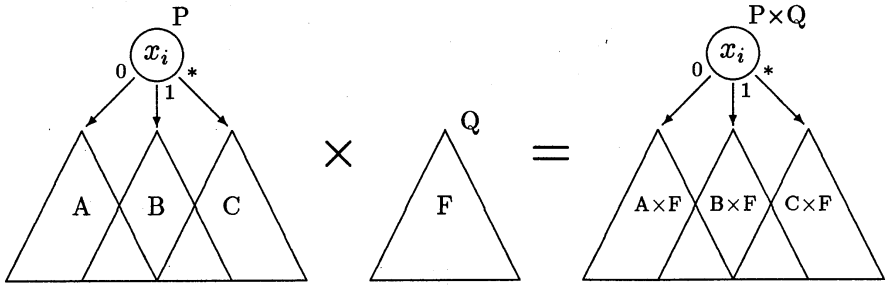
ii) **Cartesian Product**

The set operator $\times$ is used to make the product of two sum-of-products forms. For exam-

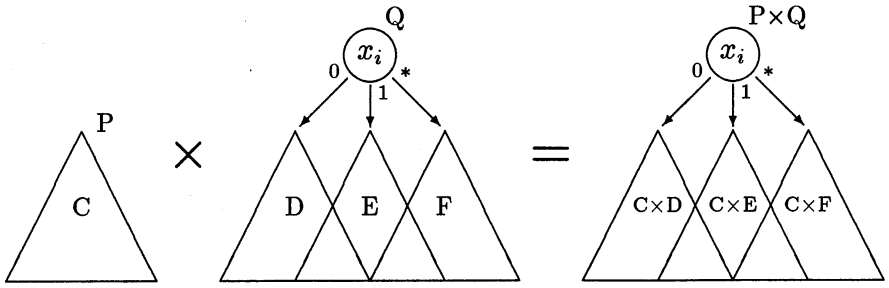
ple, $\{x_1\overline{x_2}x_3,\ x_2,\ \overline{x_2}x_3\}\times\{x_1,\ \overline{x_3}\}=\{x_1\overline{x_2}x_3,\ x_1x_2,\ x_2\overline{x_3}\}$ can be regarded as $(x_1\overline{x_2}x_3+x_2+\overline{x_2}x_3)(x_1+\overline{x_3})=x_1\overline{x_2}x_3+x_1x_2+x_2\overline{x_3}$. Figure 6 shows the procedure for the operator $\times$ on TDDs. The procedure terminates when it reaches to the leaf nodes, where the rules $P\times\emptyset=\emptyset$ and $P\times\{1\}=P$ are used.



(a) λ(node of P)=λ(node of Q)



(b) λ(node of P)>λ(node of Q)



(c) λ(node of P)<λ(node of Q)

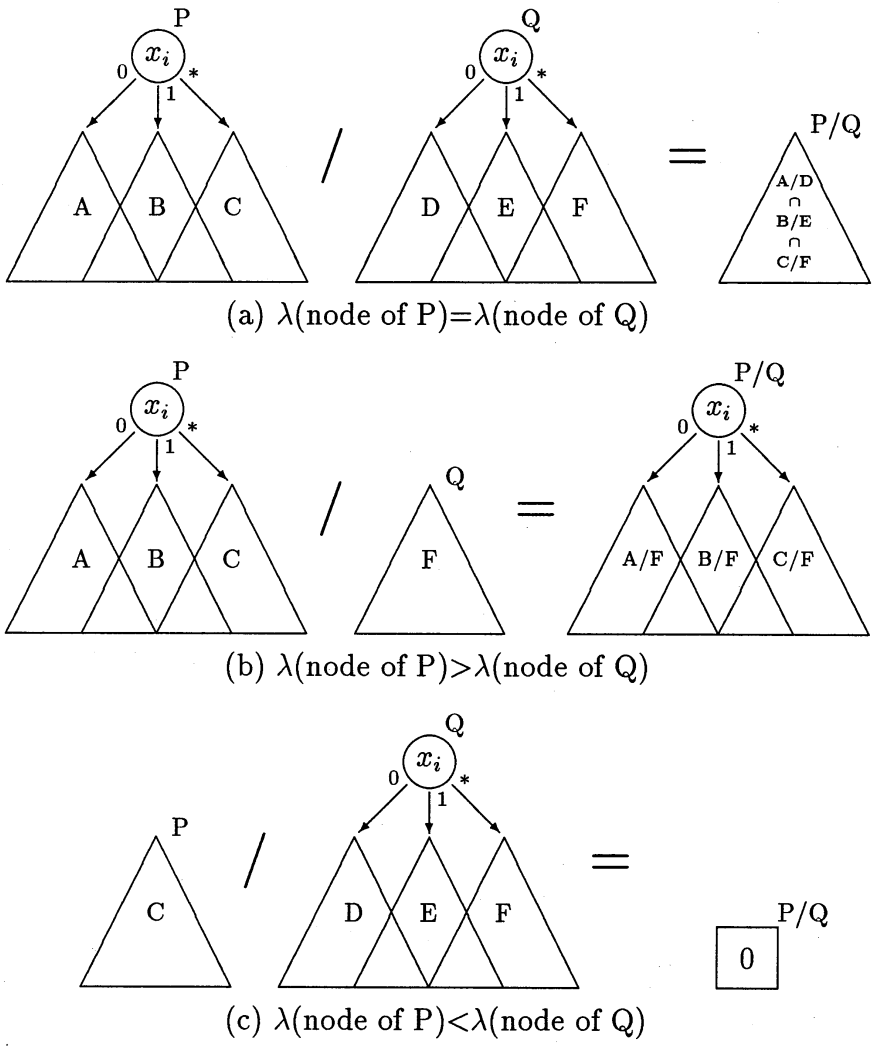
Figure 6: Procedure to compute P×Q

iii) **Weak Division**

The weak division operator / is used to get a quotient on the field of Cartesian Product. The quotient R=P/Q is the maximum set that satisfies $R\times Q\subseteq P$, where R does not include variables appeared in Q. For example, $\{x_1\overline{x_2}x_3,\ x_2,\ \overline{x_2}x_3\}/\{1,\ x_1\}=\{\overline{x_2}x_3\}$. Figure 7 shows the procedure for the operator / on TDDs. The procedure terminates when the rule $P/\emptyset=\infty$ or $P/\{1\}=P$ is used, where ∞ means the universal set which satisfies $\infty\cap R=R$ for any set R.

iv) **Intersection**

The set operator $\cap$ is used to get common products between two sets of products. The procedure for the operator $\cap$ on TDDs is similar to the one for $\cup$ save that its termination rules are $P\cap\emptyset=\emptyset$, $\emptyset\cap Q=\emptyset$, and $\{1\}\cap\{1\}=\{1\}$. It is also used within the weak division procedure.



(a) λ(node of P)=λ(node of Q)

(b) λ(node of P)>λ(node of Q)

(c) λ(node of P)<λ(node of Q)

Figure 7: Procedure to compute P/Q

v) **Difference**

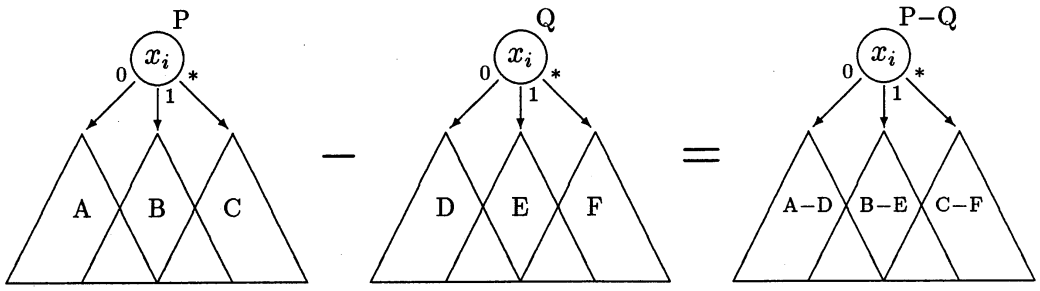
The set operator $-$ is used to remove some products from a set of products. It can be used to get a remainder after a weak division as $P-((P/Q)\times Q)$. Figure 8 shows the procedure for the operator $-$ on TDDs. The procedure terminates when it reaches to the leaf nodes, where the rules $P-\emptyset=P$, $\emptyset-Q=\emptyset$, and $\{1\}-\{1\}=\emptyset$ are used.

3.2 Manipulating ringsum-of-products forms on TDDs

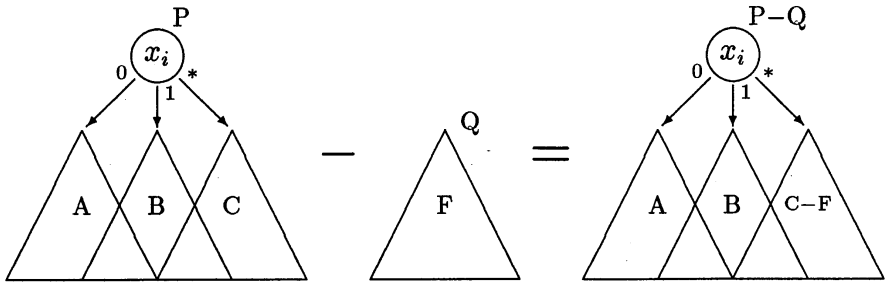
We consider essential operations on TDDs, in which we represent ringsum-of-products forms, in this section, namely, Symmetric Difference, Ringsum Product, Weak Division, and Intersection.

i) **Symmetric Difference**

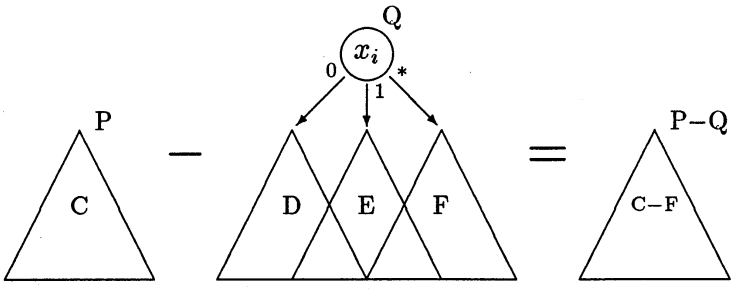
The set operator $\oplus$ is used to make the ringsum of two ringsum-of-products forms. For example, $\{x_1\overline{x_2}x_3,\ x_2,\ \overline{x_2}x_3\}\oplus\{x_2,\ \overline{x_1}x_3\}=\{x_1\overline{x_2}x_3,\ \overline{x_2}x_3,\ \overline{x_1}x_3\}$ can be regarded as $(x_1\overline{x_2}x_3\oplus x_2\oplus\overline{x_2}x_3)\oplus(x_2\oplus\overline{x_1}x_3)=x_1\overline{x_2}x_3\oplus\overline{x_2}x_3\oplus\overline{x_1}x_3$. It can be used to get a remainder after a weak division as $P\oplus((P/Q)\otimes Q)$. The procedure for the operator $\oplus$ on TDDs is similar to the one for $\cup$ in the previous section save that its termination rules are $P\oplus\emptyset=P$, $\emptyset\oplus Q=Q$, and $\{1\}\oplus\{1\}=\emptyset$.



(a) λ(node of P)=λ(node of Q)



(b) λ(node of P)>λ(node of Q)



(c) λ(node of P)<λ(node of Q)

Figure 8: Procedure to compute P−Q

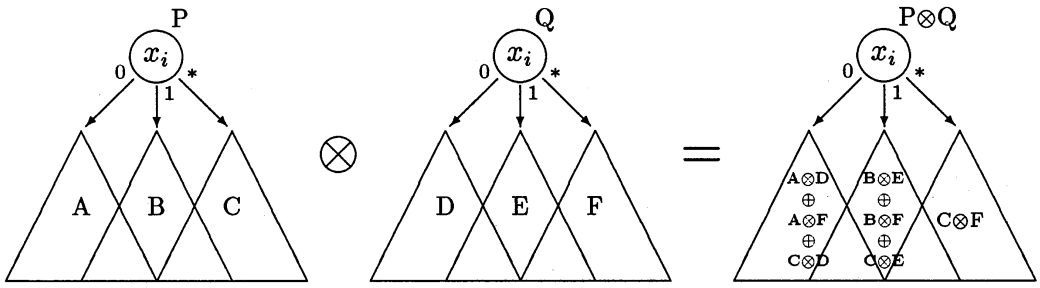
ii) **Ringsum Product**
The set operator $\otimes$ is used to make the product of two ringsum-of-products forms. For example, $\{x_1\overline{x_2}x_3,\ x_2,\ \overline{x_2}x_3\}\otimes\{x_1,\ \overline{x_3}\}=\{x_1x_2,\ x_2\overline{x_3}\}$ can be regarded as $(x_1\overline{x_2}x_3\oplus x_2\oplus\overline{x_2}x_3)(x_1\oplus\overline{x_3})=x_1x_2\oplus x_2\overline{x_3}$. Figure 9 shows the procedure for the operator $\otimes$ on TDDs. The procedure terminates when it reaches to the leaf nodes, where the rules $P\otimes\emptyset=\emptyset$ and $P\otimes\{1\}=P$ are used.

iii)-iv) **Weak Division and Intersection**
Same as those mentioned in the previous section, though we regard the quotient R=P/Q here as the maximum set that satisfies $R\otimes Q\subseteq P$, because $R\otimes Q=R\times Q$ when R does not include variables appeared in Q.

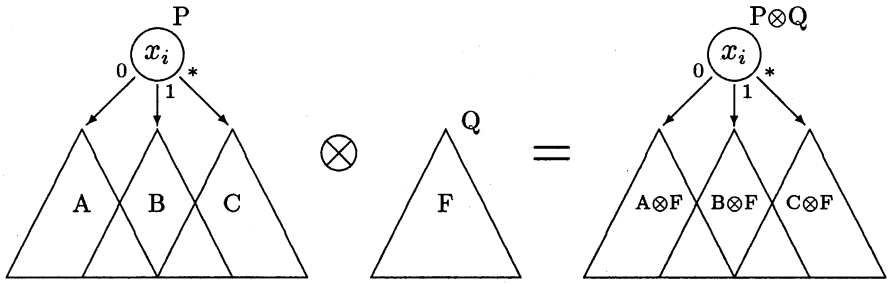
3.3 Emulating BDDs

When a sum-of-products form P is given, we can obtain the sum-of-minterms form Q which expresses the logic function expressed by P with Shannon Expansion:

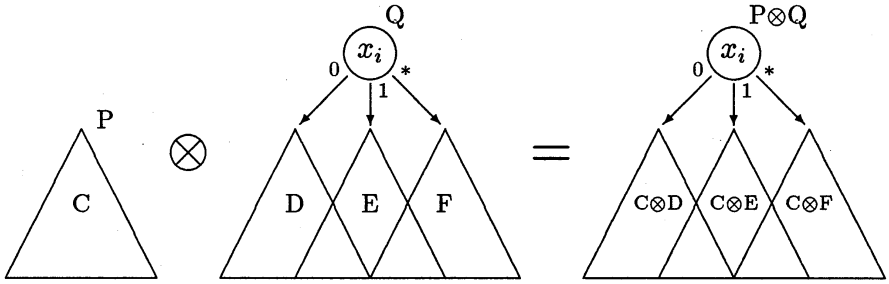
$$Q = \{\overline{x_1},\ x_1\}\times\{\overline{x_2},\ x_2\}\times\cdots\times\{\overline{x_n},\ x_n\}\times P\ .$$



(a) λ(node of P)=λ(node of Q)



(b) λ(node of P)>λ(node of Q)



(c) λ(node of P)<λ(node of Q)

Figure 9: Procedure to compute P⊗Q

When a ringsum-of-products form P is given, we can obtain the ringsum-of-minterms form Q which expresses the logic function expressed by P with Shannon-Davio Expansion:

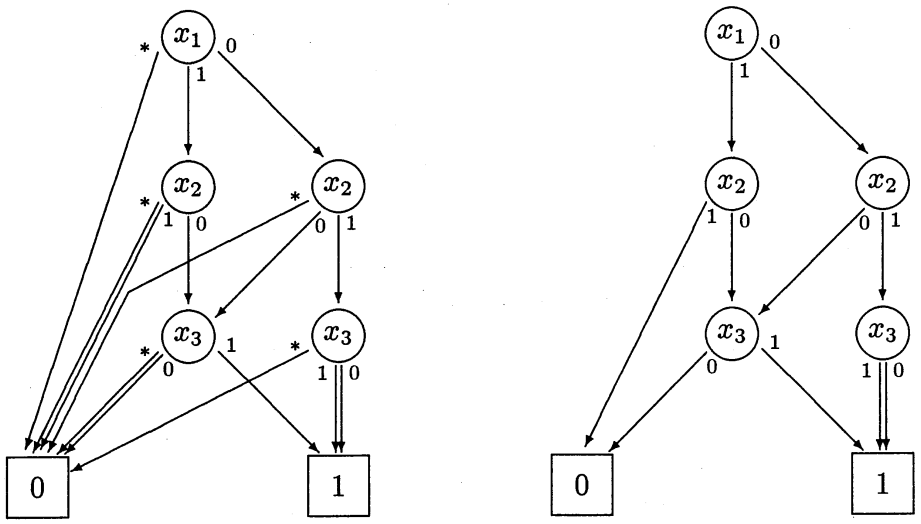
$$Q = \{\overline{x_1}, x_1\}\otimes\{\overline{x_2}, x_2\}\otimes\cdots\otimes\{\overline{x_n}, x_n\}\otimes P\ .$$

TDDs, whose source node represents a set of minterms, bear a close resemblance to quasi-reduced BDDs† whose source node represents the logic function expressed by the sum of the minterms. Since in TDDs each path from a node to $\boxed{1}$ corresponds with a product in the set represented by the node, and in quasi-reduced BDDs each path from a source node to $\boxed{1}$ corresponds with a minterm in the logic function represented by the source node. Figure 10 shows an example, in which both source nodes represent $\overline{x_1}\,\overline{x_2}x_3+\overline{x_1}x_2\overline{x_3}+\overline{x_1}x_2x_3+x_1\overline{x_2}x_3$.

Here we consider the emulation of the operations on BDDs by the operations on TDDs whose source nodes represent sets of minterms. On such TDDs, we may use all operations mentioned in the sections 3.1 and 3.2, because a set of minterms can be regarded both as a sum-of-products form and as a ringsum-of-products form.

- **Logical OR** can be emulated by the operator ∪.

†Quasi-reduced BDDs have some additional nodes to BDDs in order that any paths from the source node to $\boxed{1}$ include all variables [9].



(a) in TDDs (b) in quasi-reduced BDDs

Figure 10: Representation of a set of minterms

- **Logical EXOR** can be emulated by the operator $\oplus$.
- **Logical AND** can be emulated by the operator $\times$, $\otimes$, or $\cap$. Because $P \times Q = P \otimes Q = P \cap Q$ when P and Q are sets of minterms.
- **Logical NOT** can be emulated by $\oplus$ with the set of all minterms made of $\{\overline{x_1}, x_1\} \otimes \{\overline{x_2}, x_2\} \otimes \cdots \otimes \{\overline{x_n}, x_n\}$.
- **Restrictions** $P|_{x_i \leftarrow 0}$ and $P|_{x_i \leftarrow 1}$ can be emulated by $(P/\{\overline{x_i}\}) \times \{\overline{x_i}, x_i\}$ and $(P/\{x_i\}) \times \{\overline{x_i}, x_i\}$, respectively.

It is evident now that TDDs can emulate quasi-reduced BDDs.

4 Conclusion

We have proposed a new technique of representing sets of products named Ternary Decision Diagrams. TDDs can represent both sum-of-products forms and ringsum-of-products forms, and even can emulate BDDs. We have shown procedures to manipulate sets of products on TDDs. The procedures are very easy to implement, and very useful in the field of VLSI logic system designs. We are sure that TDDs will take the place of BDDs at implementing CAD systems in the near future.

Acknowledgement

The author would like to express his appreciation to the members of Professor Yajima's Research Laboratory of Kyoto University for their fruitful discussions. The author also thanks Mr. Minato of NTT LSI Laboratories for his beneficial advices on the operations presented in this paper.

References

[1] S. B. Akers: Binary Decision Diagrams, IEEE Transactions on Computers, Vol. C-27 (1978), pp. 509-516.

[2] R. E. Bryant: Graph-Based Algorithms for Boolean Function Manipulation, IEEE Transactions on Computers, Vol. C-35 (1986), pp. 677-691.

[3] S. Malik, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli: Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment, Proceedings of ICCAD'88 (1988), pp. 6-9.

[4] Y. Matsunaga and M. Fujita: Multi-level Logic Optimization Using Binary Decision Diagrams, Proceedings of ICCAD'89 (1989), pp. 556-559.

[5] O. Coudert and J. C. Madre: A New Graph Based Prime Computation Technique, Logic Synthesis and Optimization (edited by T. Sasao), Kluwer Academic Publishers (1992), pp. 33-57.

[6] T. Sasao: And-Exor Expressions and Their Optimization, Logic Synthesis and Optimization (edited by T. Sasao), Kluwer Academic Publishers (1992), pp. 287-312.

[7] S. Minato: Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems, Proceedings of ACM/IEEE 30th Design Automation Conference (1993), pp. 272-277.

[8] H. Ochi, K. Yasuoka, S. Yajima: Breadth-First Manipulation of Very Large Binary Decision Diagrams, Proceedings of ICCAD'93 (1993), pp. 48-55.

[9] Y. Takenaga and S. Yajima: Computational Complexity of Manipulating Binary Decision Diagrams, IEICE Transactions on Information and Systems, Vol. E77-D (1994), pp. 642-647.